

Using hough techniques to find lines and circles

Here we learn how to use opencv's hough lines

```
|[ns opencv3.hough
  (:require
    [opencv3.utils :as u]
    [opencv3.colors.rgb :as color]
    [opencv3.core :refer :all]))
```

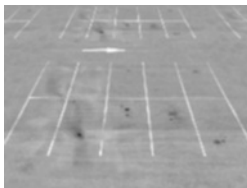
```
nil
```

Let's load the target image, and convert it to gray with a bit of blur, so that canny can work on it.

```
(def parking (-> "resources/images/lines/parking.png" imread (u/resize-by 0.3)))
(u/mat-view parking)
```

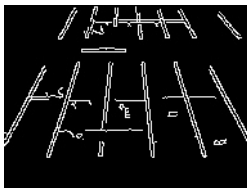


```
(def gray      (-> parking clone (cvt-color! COLOR_BGR2GRAY) (gaussian-blur! (new-size 3
3) 0 ) ))
(u/mat-view gray)
```



Edges detection is done using canny.

```
(def edges      (-> gray clone (canny! 50 150 )))
(u/mat-view edges)
```



Let's setup the parameters for the hough lines detection.

```
(def rho 1) ; distance resolution in pixels of the Hough grid
(def theta (/ Math/PI 180)) ; angular resolution in radians of the Hough grid
(def min-intersections 15) ; minimum number of votes (intersections in Hough grid cell)
(def min-line-length 50) ; minimum number of pixels making up a line
(def max-line-gap 20) ; maximum gap in pixels between connectable line segments

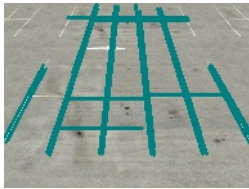
(def lines (new-mat))
(hough-lines-p edges lines rho theta min-intersections min-line-length max-line-gap)
```

```
nil
```

Now that we have the lines, let's draw them on a clone of the original picture.

```
(def result (clone parking))
(dotimes [ i (.rows lines)]
  (let [ val (.get lines i 0)]
    (line result
      (new-point (nth val 0) (nth val 1))
      (new-point (nth val 2) (nth val 3))
      color/teal
      2)))
```

```
(u/mat-view result)
```



Finally, output all the intermediate images used in this guide. Let's not forget that `hconcat` and `vconcat` need all the pictures to be in the same format, in our case, the same number of channels. We convert back `gray` and `edges` with the proper number of channels.

```
(u/mat-view (hconcat! [  
  parking  
  (-> gray clone (cvt-color! COLOR_GRAY2RGB))  
  (-> edges clone (cvt-color! COLOR_GRAY2RGB))  
  result] ))
```



Finding only pockets on a pool table

```
(def pool  
  (->  
    "https://raw.githubusercontent.com/badlogic/opencv-fun/master/data/topdown-6.jpg"  
    u/mat-from-url  
    (u/resize-by 0.5)))  
(u/mat-view pool)
```



```
(def gray (-> pool clone (cvt-color! COLOR_BGR2GRAY)))  
(def minRadius 14)  
(def maxRadius 18)  
(def circles (new-mat))  
(hough-circles gray circles CV_HOUGH_GRADIENT 1 minRadius 120 10 minRadius maxRadius)
```

```
nil
```

```
(def output (clone pool))  
(dotimes [i (.cols circles)]  
  (let [circle (.get circles 0 i) x (nth circle 0) y (nth circle 1) r (nth circle 2) p  
        (new-point x y)]  
    (opencv3.core/circle output p (int r) color/red-2 1)))  
(u/mat-view output)
```

